Draining Bytes in Flight to Reduce Queuing before TCP Slow Start Exit

Maryam Ataei Kachooei[†], Jae Chung[*], Feng Li[*], Benjamin Peters[*], Mark Claypool[†]
[†]Worcester Polytechnic Institute, Worcester, MA, USA
[*]Viasat Inc., Marlboro, MA, USA
[†]{mataeikachooei, claypool}@wpi.edu [*]{jaewon.chung, feng.li, benjamin.peters}@viasat.com

***Abstract*—TCP slow start exit often occurs after the sender surpasses the available capacity, resulting in excess in-flight data in the bottleneck queue. We propose a drain mechanism that complements slow start exit algorithms by draining excess in-flight data to a target congestion window before entering congestion avoidance. The drain gradually reduces the congestion window toward the target while still allowing new data to be transmitted. We implement and evaluate the mechanism in SEARCH, a rate-based slow start exit algorithm, in the Linux TCP stack. Evaluation across 12 network configurations (varying queue capacities and delay jitter) finds draining reduces mean excess RTT by about 40% and reduces retransmission overhead while maintaining or improving goodput utilization for larger queues. The latency benefit grows with queue size, reaching about 48% for 4× and 8×BDP queues, and retransmission overhead is halved for 4×BDP queues.**

***Index Terms*—TCP slow start, congestion control, queuing delay, congestion window, bandwidth-delay product**

I. Introduction

Many Internet transfers are short, including requests for web page objects and HTTP video chunks, making TCP slow start an important determinant in their performance [1], [2]. During slow start, the TCP sender rapidly increases its congestion window (`cwnd`) to quickly reach the available network bandwidth. However, slow start typically only exits after exceeding the available bandwidth, resulting in buildup at the bottleneck queue, increasing latency not only for the transmitting flow but also for other flows sharing the same path [3].

Traditional slow start continues exponential growth of the cwnd until it reaches the slow start threshold (`ssthresh`), which often unfortunately results in packet loss when the bottleneck queue fills and drops incoming packets [4], [5]. To avoid this, several mechanisms have been proposed to detect when slow start should end before the bottleneck queue fills. HyStart uses increases in round-trip time (RTT) and acknowledgment-train (ACK-train) measurements to identify congestion [6]. HyStart++ refines HyStart's delay-based detection and adds a conservative slow start phase of reduced cwnd growth rate [7] in an attempt avoid premature slow start exits. Slow start Exit at Right Chokepoint (SEARCH) uses the divergence between sent and delivered data to detect when the available bandwidth has been reached [8].

Although these mechanisms use different congestion signals, their slow start exit decisions rely on feedback that is delayed relative to the sender's transmission time by at least a round-trip time. During this time, exponential congestion window growth can cause the sender to exceed the available network capacity, accumulating excess in-flight data and increasing queuing delay. Thus, even when an exit mechanism detects saturation before loss, excess data has typically already bloated the bottleneck queue. Detecting saturation and managing the resulting excess in-flight data are therefore distinct challenges.

Existing approaches handle the transition from slow start to congestion avoidance differently. HyStart and HyStart++ primarily control when exponential growth ends and do not explicitly drain excess in-flight data [6], [7]. In contrast, BBR incorporates a drain phase following Startup during which a reduced pacing rate brings in-flight data toward the estimated bandwidth-delay product (BDP) [9]. However, BBR's drain phase is integrated into its model-based congestion control design. This motivates a drain mechanism that can complement existing slow start exit algorithms without requiring changes to their saturation detection logic.

In this paper, we propose a controlled drain mechanism that operates between slow start saturation detection and the transition to congestion avoidance. Once saturation is detected, the mechanism estimates a target congestion window from observed delivery behavior and gradually adjusts the congestion window toward that target to limit new transmissions and allow excess outstanding data to decrease as acknowledgments arrive. The mechanism is decoupled from the saturation detection logic and can potentially complement different slow start exit algorithms, provided the required sender-side measurements are available.

We implement the proposed drain mechanism using SEARCH [8] in the Linux TCP stack and compare its performance with SEARCH without drain under varying queue capacities and delay jitter levels. Performance evaluation focuses on queuing delay, congestion window behavior, retransmissions, and data delivery to illuminate the benefits and tradeoffs of controlled draining.

The remainder of this paper is organized as follows: Section II reviews related work; Section III presents the drain mechanism; Section IV describes the implementation and experimental methodology; Section V presents the evaluation; and Section VI summarizes our conclusions and presents possible future work.

II. Related Work

A. Slow Start Exit Mechanisms

Slow start increases the TCP congestion window (cwnd) exponentially until the cwnd reaches ssthresh [4], but since the initial threshold often does not reflect the available network capacity, slow start often exits after the bottleneck queue has saturated and packets are lost.

Several mechanisms detect approaching congestion before packet loss. HyStart uses RTT increases and ACK-train measurements [6] to detect congestion, while HyStart++ refines delay-based detection and introduces Conservative Slow Start (CSS) to slow growth and prevent premature slow start exits [7]. SUSS accelerates initial window growth while using delay-based detection for slow start exit [10]. Limited Slow-Start reduces window growth beyond a configured threshold but does not adapt this limit to path capacity [11].

Other approaches use delivery information as a congestion signal: BBR monitors delivery rate growth during Startup [9], while SEARCH compares sent and delivered data over measurement windows [8].

These mechanisms primarily determine when to end or modify the exponential cwnd growth in slow start. However, even improvements to the slow start exit mechanisms have feedback delay, meaning excess data accumulates in the bottleneck queue before network capacity saturation is detected, making the management of queue build up a separate concern.

B. Managing Excess In-Flight Data

Loss-based congestion recovery reduces the congestion window, but does so in response to packet loss (e.g., from a saturated router queue). Previous SEARCH work (version 3.0) [12] considered an optional congestion window rollback following saturation detection. However, reducing the congestion window in this manner restricts new transmissions, leaving the approach vulnerable to timeouts. Our revised approach instead adjusts the congestion window gradually toward an estimated target during a controlled drain phase.

BBR also addresses excess in-flight data through a drain state following the initial Startup state. During the drain phase, BBR reduces its pacing rate to bring in-flight data toward the estimated BDP [9]. However, BBR's draining behavior is integrated with its bandwidth estimation, pacing, and congestion control state machine, making it difficult to generalize to other TCP approaches. In contrast, our mechanism is decoupled from saturation detection logic and is designed to complement existing slow start exit algorithms.

C. Window Reduction Mechanisms

Other congestion control mechanisms also regulate outstanding data. Proportional Rate Reduction (PRR) controls transmissions during loss recovery as the sender approaches a reduced congestion window [13]. Congestion Window Validation adjusts the window following application-limited periods when its previous value may be stale [14]. Careful Resume includes mechanisms for reducing the sending window when congestion is detected after resuming with previously learned path information [15].

Unlike these mechanisms, our work focuses on controlled draining triggered by slow start saturation detection, before completing the transition to congestion avoidance.

D. Network Queue Management

Active Queue Management (AQM) mechanisms, such as CoDel, reduce persistent queuing delay by dropping or marking packets at the bottleneck [3], [16]. These approaches complement sender-side congestion control but require support within the network. In contrast, our proposed mechanism operates entirely at the sender without requiring AQM in intermediate devices.

III. Drain Design

We propose a controlled drain mechanism that operates between slow start saturation detection and the transition to congestion avoidance. The mechanism is decoupled from the saturation detection logic and uses sender-side delivery history to estimate a target congestion window. It also uses the congestion window, outstanding data, and acknowledgment information to control the drain phase.

Following saturation detection, the sender enters a controlled drain phase rather than immediately reducing the congestion window to the estimated target. During this phase, the congestion window is decreased gradually as outstanding data is acknowledged, allowing excess in-flight data to decrease while still transmitting new data before transitioning to congestion avoidance.

The mechanism can complement different slow start exit algorithms provided that a saturation signal and the required sender-side measurements are available. In our implementation, SEARCH supplies the saturation detection signal, while drain computes its target using the delivery history.

A. Target Congestion Window

The drain mechanism estimates a congestion window W_{BDP} from recently delivered data measured over a window whose duration is determined by the initial RTT. To account for estimation uncertainty, the drain target is defined as $W_d = \gamma W_{\mathrm{BDP}}$, where $\gamma \geq 1$ provides a configurable margin above the estimated BDP. A larger γ retains a greater margin, while a smaller value brings the target closer to the estimate. Our implementation uses $\gamma = 1.25$.

Figure 1 presents preliminary BDP-estimation measurements collected during the development of the drain mechanism. The graph shows the distribution of the estimated window W_{BDP} normalized by the actual BDP across 500 runs over a range of network conditions. In the graph, a normalized value of 1 indicates that the estimate matches the BDP; values above 1 are an overestimate, and values below 1 are an underestimate. A normal distribution fitted to the measurements has mean $\mu = 1.01$ and standard deviation $\sigma = 0.25$. The margin $\gamma = 1.25$ corresponds to about one standard deviation above the mean estimate ($\mu + \sigma = 1.26$).

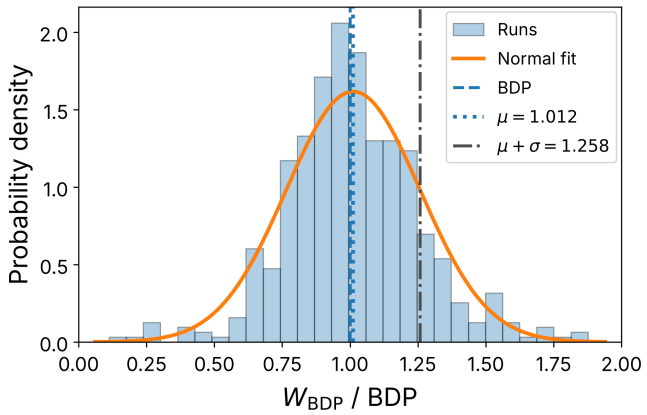

Fig. 1: Distribution of the estimated BDP window normalized by the configured BDP, with the fitted normal distribution.

With this margin, the target W_d is at or above the BDP in 83% of the runs, which is preferable to a target below the BDP.

B. Drain Initiation

Following saturation detection, the sender determines whether draining is needed based on the current congestion window W, outstanding data F, and drain target W_d. Draining begins only when $W > W_d$ and $F > W_d$. The first condition ensures that the congestion window exceeds the target, while the second ensures that outstanding data also exceeds it. These conditions may not hold following congestion window reductions, during loss recovery, or under application-limited transmission. If either condition is not satisfied, the sender skips draining and completes the transition to congestion avoidance without an additional window reduction.

C. Controlled Draining

During the drain phase, the congestion window is adjusted based on the outstanding data and acknowledged segments rather than following the exponential growth of slow start.

Let A represent an accumulator of acknowledged segments and let K denote the drain rate parameter. For each acknowledgment, the accumulator is updated as $A \leftarrow A + N_{\mathrm{acked}}$, where N_{acked} is the number of newly acknowledged segments.

The sending allowance is computed as

$$C = \left\lfloor \frac{A}{K} \right\rfloor, \qquad A \leftarrow A \bmod K.$$

The drain rate K controls the sending allowance. Smaller values permit more transmissions, while larger values restrict them.

The congestion window is then adjusted according to

$$W \leftarrow \max(F + C,\ W_d),$$

where F represents the current outstanding data in segments.

This adjustment relates the congestion window to the outstanding data while providing a limited sending allowance based on acknowledged segments. As outstanding data decreases, the congestion window can progressively approach the drain target without an immediate rollback.

D. Drain Completion and Loss Recovery

The drain phase completes when the congestion window reaches the drain target. Under normal operation, the slow start threshold is then set to the target congestion window, and the sender transitions to congestion avoidance.

Loss recovery may overlap with the drain phase. The drain logic checks whether the congestion window has already reached or fallen below the drain target before applying further drain adjustments. This prevents the drain mechanism from continuing to reduce the window after recovery has brought it to or below the target. When recovery is active, the congestion control response to loss must retain priority over the drain adjustment.

IV. Evaluation

A. Implementation

We implement the proposed drain mechanism in the Linux TCP stack as an extension to TCP Cubic with SEARCH. The implementation supports SEARCH with and without draining, allowing us to isolate the effect of the drain mechanism while keeping the underlying congestion control implementation unchanged.

With drain enabled, when SEARCH detects saturation, instead of immediately exiting slow start to enter congestion avoidance, it estimates a target congestion window and initiates the controlled drain phase described in Section III. The implementation uses a drain rate of $K = 16$ (i.e., decrease 15 packets for every 16 packets received).

B. Testbed

Our testbed consists of a virtual machine acting as the TCP sender and a laptop acting as the TCP receiver, connected through a Raspberry Pi router that emulates a configurable network bottleneck. This setup enables controlled evaluation of different network conditions while keeping the sender and receiver endpoints fixed.

The sender runs Debian GNU/Linux 13 with a modified Linux 6.13.3 kernel supporting SEARCH and the proposed drain mechanism. For performance comparison, the baseline variant uses SEARCH without the drain phase.

The receiver is a laptop equipped with an Intel Core Ultra 7 155U processor and 15 GB of RAM, running Ubuntu 24.04.2 LTS with an unmodified Linux 6.13.3 TCP stack.

The router is a Raspberry Pi with a quad-core ARM Cortex-A72 processor, running Linux 5.10.63-v7l+, also with an unmodified TCP stack. It uses Linux traffic control (`tc`) to configure the bottleneck bandwidth, propagation delay, queue capacity, and jitter.

C. Network Conditions

We evaluate the drain mechanism under different network conditions to examine queuing behavior during slow start. The bottleneck bandwidth is fixed at 100 Mb/s, with a baseline RTT of 100 ms. We vary the router buffer capacity across 2, 4, and 8 times the BDP, and the jitter across 0, 26, 50, and 100 ms, corresponding to zero, one-quarter, one-half, and one baseline RTT, respectively. These configurations allow us to investigate how buffer capacity and network delay variability affect congestion window overshoot, queuing delay, and the effectiveness of the proposed drain mechanism.

D. Evaluation Metrics

Our evaluation focuses on latency, congestion behavior, and data delivery during slow start and immediately after slow start exit.

To focus on the effects of slow start and its immediate aftermath, we analyze the interval $[0, 2T_s]$, where T_s denotes the time of slow start exit measured relative to the connection start. This interval captures both the slow start phase and the subsequent behavior, allowing us to examine the effects of draining on queuing delay, congestion window evolution, retransmissions, and data delivery.

We consider the following metrics:

- **Excess RTT** – the difference between the observed RTT and the configured base RTT (i.e., the minimum link latency). We evaluate mean excess RTT over $[0, 2T_s]$. Because the configured network includes jitter, excess RTT reflects both queuing delay induced by slow start and delay variation added by the network configuration.
- **Congestion window behavior** – the congestion window normalized by the true path BDP, used to examine its evolution around slow start exit and during the drain phase.
- **Retransmissions** – the number of retransmitted TCP segments converted to bytes using the corresponding maximum segment size (MSS), and the retransmitted bytes reported as a percentage of acknowledged bytes. This normalizes retransmission overhead by the amount of useful data delivered.
- **Goodput utilization** – the amount of acknowledged data divided by the duration of the evaluation window, normalized by the configured bottleneck bandwidth.
- **Drain target** – the ratio of the selected drain target congestion window to the true path BDP at drain initiation.
- **Data transfer threshold time** – the elapsed time required for cumulative acknowledged data to increase by specified amounts between 0.1 and 20 MB. These thresholds are measured during continuous transfers rather than independent object downloads.

E. Experimental Procedure

We compare two configurations: SEARCH without draining and SEARCH with the proposed drain mechanism (SEARCH+Drain). Both use the same saturation detection algorithm, allowing us to isolate the effects of draining on queuing delay, congestion window behavior, and retransmissions.

For each network configuration, we conduct 30 independent `iperf3` transfers for each variant, with each transfer lasting 10 seconds. We evaluate all combinations of buffer sizes and jitter levels while keeping the bottleneck bandwidth and baseline RTT fixed.

We collect kernel-level congestion control logs to analyze saturation detection, congestion window evolution, outstanding data, drain initiation and completion, and retransmissions. We also record RTT samples and cumulative acknowledged bytes to evaluate latency, goodput, and data transfer threshold times.

V. Results

We compare SEARCH with and without drain to evaluate the impact of controlled draining on queuing delay, congestion window behavior, retransmission overhead, and data delivery across different network conditions, as well as the sensitivity to the drain rate.

A. Drain Behavior and Target Selection

Figure 2 illustrates a representative session with a 4×BDP queue and 50 ms jitter. Without drain (top left graph), slow start exits after about a second, but the window continues to grow to over 4×BDP. With drain (right graph), slow start begins draining after about a second (green line), and the window decreases toward a target of 1.02×BDP which it reaches about about 1.5 seconds. The RTT behavior reflects these behaviors. Without drain, the RTT rises to about 450 ms and remains there during congestion avoidance. With drain, the RTT continues to increase after draining begins and peaks at 511 ms, then drops within 30 ms to near the base RTT (about 120 ms). This delayed response occurs because ACK-based RTT measurements reflect packets transmitted earlier, including packets that encountered the queue before drain began. Once these packets are acknowledged, RTT samples come from packets sent during the drain, which encountered a much shorter queue. Thus, while reducing the congestion window does not immediately reduce the measured RTT, it limits the persistence of elevated queuing delay.

Table I reports the drain target normalized by the true path BDP at drain initiation, based on 322 testbed runs. Without jitter, the selected target is consistent, remaining near 1.27×BDP across all three queue capacities, with standard deviations below 0.01. With jitter, the target varies a bit more – for example, at 100 ms jitter, the mean target ranges from 0.87×BDP for the 2×BDP queue to 1.28×BDP for the 4×BDP queue. These results indicate that the target is relatively stable under constant-delay conditions, while network jitter increases the range of congestion window drain targets.

B. Queuing Delay

Figure 3 compares the mean excess RTT, computed as the difference between observed RTT and the configured base RTT (100 ms), showin with 95% confidence intervals. For each

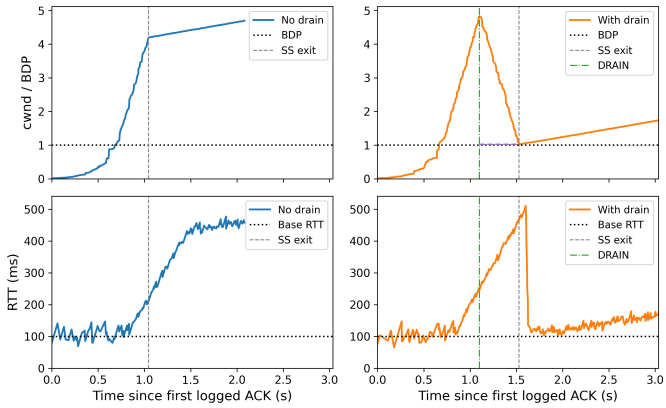

Fig. 2: **Left: without drain, Right: with drain.** Representative runs showing congestion window (top) and RTT (bottom) for a 4×BDP queue with 50 ms jitter. The green line marks the start of draining; with drain, slow start exit (gray) marks its end.

TABLE I: Drain target normalized by true BDP at drain initiation (mean ± standard deviation). Numbers in parentheses indicate sample count N.

	Jitter (ms)			
Queue	0	26	50	100
2×BDP	1.27 ± 0.01 (30)	1.16 ± 0.29 (26)	1.07 ± 0.33 (23)	0.87 ± 0.54 (18)
4×BDP	1.27 ± 0.01 (30)	1.13 ± 0.37 (30)	1.29 ± 0.69 (30)	1.28 ± 0.69 (24)
8×BDP	1.27 ± 0.01 (30)	1.17 ± 0.23 (30)	1.29 ± 0.30 (30)	1.10 ± 0.73 (26)

queue capacity, results are aggregated across the four jitter settings.

For the 2×BDP queue, SEARCH without drain and SEARCH with drain have similar mean excess RTT (approximately 141 ms and 134 ms). For larger queues, the difference is substantial: mean excess RTT decreases from approximately 233 ms to 120 ms for the 4×BDP queue and from approximately 269 ms to 139 ms for the 8×BDP queue, a reduction of about 48% in both cases.

Without drain, mean excess RTT grows with queue capacity, as excess in-flight data remains in the larger queue after slow start exit. With drain, mean excess RTT stays between 120 and 139 ms for all three queue capacities. For the 2×BDP queue, losses already limit the standing queue for both variants, leaving little excess data to drain.

C. Retransmission Overhead

Figure 4 compares retransmissions, expressed as retransmitted bytes relative to acknowledged bytes, aggregated across the four jitter settings for each queue capacity. For the 2×BDP queue, drain reduces retransmission overhead modestly, from 24% to 21%, but more substantially for the 4×BDP queue, from 9.1% to 4.5%. For the 8×BDP queue, retransmission

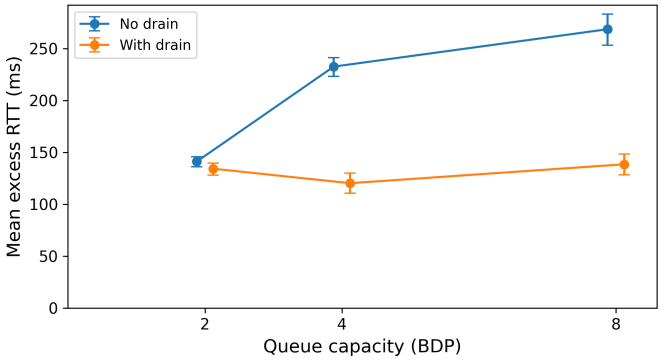

Fig. 3: Mean excess RTT without and with drain across three queue capacities, aggregated over the four jitter settings. Error bars show 95% confidence intervals. Excess RTT is measured relative to the configured 100 ms base RTT.

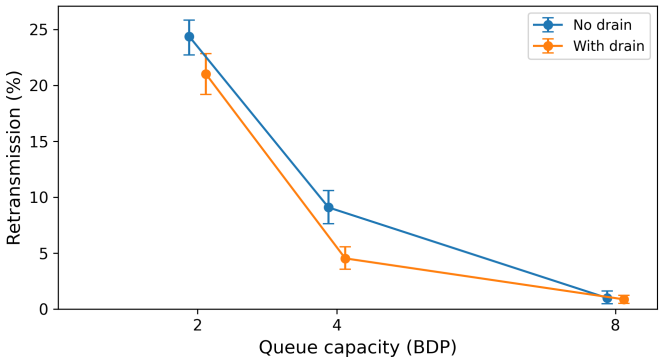

Fig. 4: Retransmission overhead without and with drain, across three queue capacities, aggregated over the four jitter settings. Error bars show 95% confidence intervals. Metrics are computed over $[0, 2T_s]$.

overhead is already low without draing (about 1%) so drain reduces it little.

Figure 5 combines retransmission overhead and mean excess RTT, with one point per queue capacity, the bars showing 95% confidence intervals both horizontally and vertically. Without drain, larger queues trade retransmissions for delay: from the 2× to the 8×BDP queue, retransmission overhead falls from 24% to 1%, while mean excess RTT rises from 141 to 269 ms. With drain, retransmission overhead follows the same trend, but mean excess RTT stays between 120 and 139 ms, so drain removes most of the delay that larger queues allow to build up.

Drain also does not reduce goodput utilization, computed as the average acknowledged data rate over $[0, 2T_s]$, normalized by the configured bottleneck bandwidth (100 Mb/s). For $K = 16$ (Table II), goodput utilization is nearly unchanged for the 2×BDP queue (62% and 63%) and increases from 58% to 69% for the 4×BDP queue and from 67% to 72% for the 8×BDP queue.

Overall, controlled draining reduces delay and retransmis-

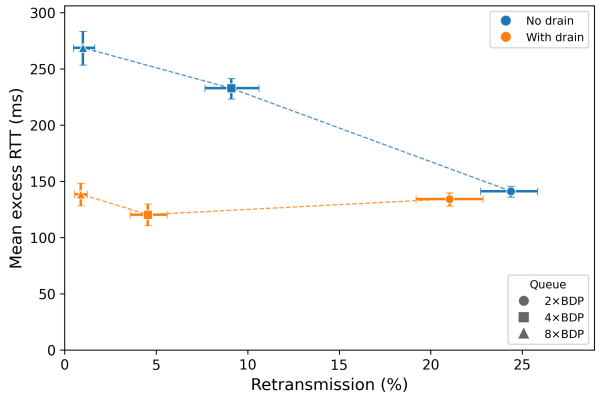

Fig. 5: Retransmission overhead versus mean excess RTT without and with drain ($K = 16$), one point per queue capacity (indicated by marker shape). Results are aggregated over the four jitter settings. The error bars show 95% confidence intervals in both dimensions.

sion overhead without sacrificing goodput. The largest combined benefit occurs for the 4×BDP queue, where both mean excess RTT and retransmission overhead are roughly halved.

D. Data Delivery Across Transfer Sizes

To examine the effect of drain on data delivery, we measure the elapsed time required to acknowledge specified amounts of data, ranging from 0.1 to 20 MB. These measurements are obtained from cumulative acknowledged-byte counters during continuous transfers, rather than from independent object downloads.

Figure 6 presents the mean elapsed time required to reach each acknowledged-data threshold, aggregated across all 12 network configurations separately with and without drain. For larger thresholds, both with and wihtout drain exhibit similar completion times, but for larger thresholds, drain generally achieves slightly lower completion times, with the difference becoming more visible at 12–20 MB.

These observations suggest that the reductions in RTT and retransmission overhead achieved by drain do not come at the expense of data delivery and, in fact, the reduction in retransmission improves utilization.

E. Sensitivity to Drain Rate

The drain rate determines how many new segments are transmitted during draining. Up to now, the drain rate was set to K = 16, meaning 1 new segment is transferred for every 16 acknowledged segments. We evaluate drain rates K = 4, 8, 16, and 32 across the same 12 network configurations, with 30 runs per configuration for each value of K. Table II summarizes the results.

For each acknowledged segment, outstanding data decreases by $(1 - 1/K)$ segments. A smaller K therefore lengthens the drain: in the 8×BDP queue, where most drains complete, the median completed drain lasts 535 ms with $K = 4$, compared with 449 ms with $K = 16$ and 434 ms with $K = 32$. Slower

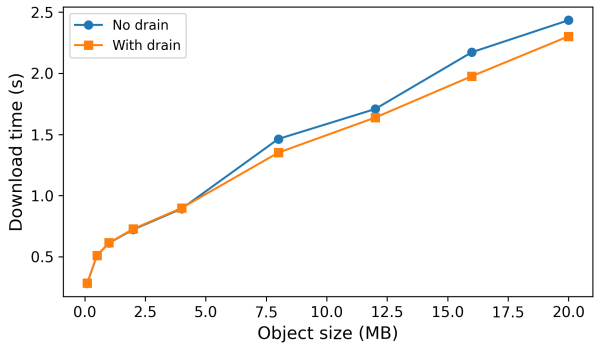

Fig. 6: Mean elapsed time to reach acknowledged data thresholds from 0.1 to 20 MB, measured relative to the start of the connection.

TABLE II: Sensitivity to the drain rate K. Means over $[0, 2T_s]$, aggregated over the four jitter settings.

Metric	Queue	With drain, K				No drain
		4	8	16	32	
Excess RTT (ms)	2×	137	137	134	131	141
	4×	136	119	120	129	233
	8×	144	138	139	132	269
Retrans. (%)	2×	21.1	20.3	21.0	20.0	24.4
	4×	5.0	4.1	4.5	5.3	9.1
	8×	0.6	0.7	0.9	0.7	1.0
Goodput (%)	2×	61.5	62.6	62.8	62.4	62.0
	4×	71.9	70.5	69.1	69.8	58.3
	8×	75.0	73.8	72.4	72.5	67.1

draining also keeps the bottleneck queue occupied longer, so loss recovery more often interrupts the drain (and starts congestion avoidance) before it reaches the target. This effect appears in the 4×BDP queue, where 16% of drains complete with $K = 4$, compared with 44% with $K = 16$ and 53% with $K = 32$. With the 2×BDP queue, loss interrupts about 90% of drains for all values of K, while with the 8×BDP queue, about 90% of drains complete for all values of K.

Despite these differences, performance changes little with K. Mean excess RTT remains between 119 and 144 ms for all values of K, compared with 233 and 269 ms for SEARCH without drain in the 4× and 8×BDP queues. Retransmission overhead and goodput utilization have overlapping 95% confidence intervals for all K, and all K retain gains over SEARCH without drain in the 4× and 8×BDP queues.

This limited effect follows from the drain design: the drain rate only limits new transmissions, while data already queued at drain initiation still leaves at the bottleneck rate. Overall, the benefits of draining are relatively insensitive to the drain rate K chosen. The results in Subsections V-A to V-D use $K = 16$.

VI. Conclusion and Future Work

TCP slow start quickly increases the congestion window to make full use of the available bandwidth before exiting to congestion avoidance, but frequently overshoots, leaving excess bytes in flight in the bottleneck queue. We propose a drain mechanism to reduce excess outstanding data and

thus queuing delay following slow start saturation detection. The mechanism estimates a target congestion window and gradually adjusts the congestion window toward it before transitioning from slow start to congestion avoidance. We implemented the mechanism in Linux TCP using SEARCH and evaluated its performance across different queue capacities and jitter conditions.

Experimental results show that drain reduces mean excess RTT by about 40% and reduces retransmission overhead relative to no draining. The benefits are particularly pronounced for larger queues: drain reduces mean excess RTT by about 48% for $4\times$ and $8\times$BDP queues and halves retransmission overhead for $4\times$BDP queues, while generally maintaining or improving goodput utilization – albeit, the effectiveness of draining depends on network conditions, particularly queue capacity and jitter. A sensitivity analysis shows that these benefits are relatively the same for drain rates K from 4 to 32.

Future work could investigate adaptive selection of the drain target and drain rate to improve robustness under varying network conditions. Evaluation over real-world paths, including Wi-Fi and satellite networks, and investigation into the applicability to other slow start exit algorithms are also possible future work.

REFERENCES

[1] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan, "Reducing Web Latency: The Virtue of Gentle Aggression," in *Proceedings of the ACM SIGCOMM Conference*, Hong Kong, China, Aug. 2013, pp. 159–170.

[2] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis, "Increasing TCP's Initial Window," Internet Engineering Task Force, RFC 6928, Apr. 2013.

[3] J. Gettys and K. Nichols, "Bufferbloat: Dark Buffers in the Internet," *Communications of the ACM*, vol. 55, no. 1, pp. 57–65, Jan. 2012.

[4] V. Jacobson, "Congestion Avoidance and Control," in *Proceedings of the ACM SIGCOMM Symposium on Communications Architectures and Protocols*, Stanford, CA, USA, Aug. 1988, pp. 314–329.

[5] L. Xu, S. Ha, I. Rhee, V. Goel, and L. Eggert, "CUBIC for Fast and Long-Distance Networks," Internet Engineering Task Force, RFC 9438, Aug. 2023.

[6] S. Ha and I. Rhee, "Taming the Elephants: New TCP Slow Start," *Computer Networks*, vol. 55, no. 9, 2011.

[7] P. Balasubramanian, Y. Huang, and M. Olson, "HyStart++: Modified Slow Start for TCP," Internet Engineering Task Force, RFC 9406, May 2023.

[8] M. A. Kachooei, J. Chung, F. Li, B. Peters, J. Chung, and M. Claypool, "Improving TCP Slow Start Performance in Wireless Networks with SEARCH," in *Proceedings of the IEEE 25th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2024.

[9] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Communications of the ACM*, vol. 60, no. 2, pp. 58–66, 2017.

[10] M. Arghavani, H. Zhang, D. Eyers, and A. Arghavani, "SUSS: Improving TCP Performance by Speeding Up Slow-Start," in *Proceedings of the ACM SIGCOMM Conference*, Sydney, Australia, Aug. 2024.

[11] S. Floyd, "Limited Slow-Start for TCP with Large Congestion Windows," Internet Engineering Task Force, RFC 3742, Mar. 2004.

[12] J. W. Chung, M. A. Kachooei, F. Li, and M. Claypool, "SEARCH – a New Slow Start Algorithm for TCP and QUIC," Internet Engineering Task Force, Internet-Draft draft-chung-ccwg-search-07, 2025, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-chung-ccwg-search/07/

[13] M. Mathis, N. Cardwell, Y. Cheng, and N. Dukkipati, "Proportional Rate Reduction (PRR)," Internet Engineering Task Force, RFC 9937, Dec. 2025, obsoletes RFC 6937.

[14] G. Fairhurst, A. Sathiaseelan, and R. Secchi, "Updating TCP to Support Rate-Limited Traffic," Internet Engineering Task Force, RFC 7661, Oct. 2015.

[15] N. Kuhn, E. Stephan, G. Fairhurst, R. Secchi, and C. Huitema, "Careful Resume: Convergence of Congestion Control from Retained State," Internet Engineering Task Force, RFC 9959, May 2026.

[16] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, "Controlled Delay Active Queue Management," Internet Engineering Task Force, RFC 8289, Jan. 2018.